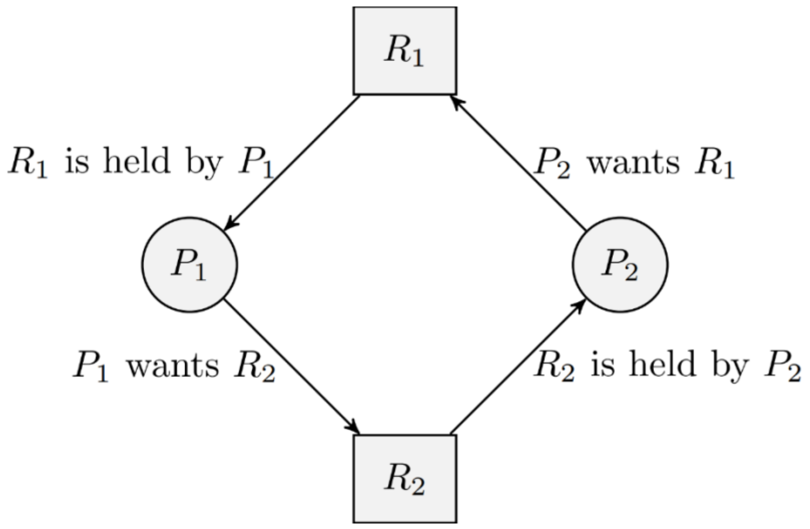
11. Sarac

Program Name: Sarac.java **Input File: sarac.dat**

Sarac is learning about Mutual Exclusion and Deadlocks which are common problems in Multi-Threaded programs and an issue that Operating System Engineers are constantly forced to address. For a resource to be considered "Mutually Exclusive", then at most one process or program is allowed to access that resource at any single point in time. In other words, no two processes can share the resource at the same time, but multiple processes may hold the resource independently over a period of time.

Since a single process may be interested in holding multiple such shared resources that need to be accessed mutually exclusively at a single instance of time, it is possible for a "Deadlock" to occur. Consider the example graph below – since both P_1 and P_2 are waiting on a resource held by the other, and neither is able to release their resource until they have obtained all resources they are interested in, a Deadlock has formed since neither process can continue.



One such way of modeling these systems is through a structure known as a Resource Allocation Graph (RAG). This is a bipartite graph where a directed edge from a resource R, to a process P, exists if and only if resource R is held by process, P. Similarly, there exists a directed edge from a process P, to a resource R, if and only if process P is interested in holding resource R. No other edges exist in the graph outside of these two types. Using this model, Sarac is able to detect the occurrence (or absence) of Deadlocks in a given system. Given a RAG, help Sarac detect the occurrence or absence of a Deadlock in the RAG.

Input: The first line of input will consist of a single integer n, $1 \leq n \leq 20$, denoting the number of test cases to follow. Each test case will begin with a line denoting two space-separated integers V, $4 \leq V \leq 10^5$, denoting the number of vertices and E, $2 \leq E \leq 10^5$, denoting the number of edges in the RAG. The next line will consist of V space-separated strings denoting the labels of V vertices in the graph, each of which will either be in the form of "`Pi`" or "`Ri`", where `i` is some integer $1 \leq i \leq V$. The next line will consist of E space-separated edges, each of which will either be in the form of "`Pi->Ri`" indicating that `Pi` is interested in holding `Ri`, or "`Ri->Pi`" indicating that `Ri` is held by `Pi`. It is guaranteed that all labels among the edges exist among the V vertices provided, and all edges adhere to the rules described above.

Output: For each of Sarac's n requests, either print the string "`Deadlock free; all is well`" if the RAG does not contain a Deadlock or the string "`Deadlock exists; not good...`" if the RAG contains a Deadlock.

~ ***Sample input and output on next page*** …~

~ ***Sarac continued*** …~

Sample input:
```
4
4 4
P1 P2 R1 R2
R1->P1 P1->R2 R2->P2 P2->R1
4 3
P1 P2 R1 R2
R1->P1 P1->R2 P2->R1
5 4
P1 P2 P3 R1 R2
R1->P1 R2->P2 P3->R1 P3->R2
7 9
R1 P1 R2 P2 R3 P3 R4
R1->P1 R2->P2 R3->P3 P1->R2 P1->R3 P2->R1 P2->R3 P3->R1 P3->R4
```

Sample output:
```
Deadlock exists; not good...
Deadlock free; all is well
Deadlock free; all is well
Deadlock exists; not good...
```